IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Patent Application

Appellants:	Bala et al.	Confirmation No.:	7644
Application No.:	09/874,170	Group Art Unit:	2123
Filed:	June 04, 2001	Examiner:	Proctor, J.

For: Networked Client-Server Architecture for Transparently Transforming and Executing Applications

APPEAL BRIEF

Table of Contents

I. Real Party in Interest

The assignee of the present application is Hewlett-Packard Development Company, L.P.

II. Related Appeals and Interferences

There are no related appeals or interferences known to the Appellant.

III. Status of Claims

Claim 1 has been previously canceled. Claims 2-24 remain pending. Claims 2-24 are rejected. This Appeal involves Claims 2-24.

IV. Status of Amendments

All proposed amendments have been entered. An amendment subsequent to the Final Action has not been filed.

V. Summary of Claimed Subject Matter

Independent Claims 2, 9, 14, 18 and 23 of the present application pertain to networked client-server architecture for transparently transforming and executing applications.

At least one embodiment of a networked system of Claim 2 is depicted at least in Figure 1 and 3. Figure 1 includes a network 16 described at least at page 11 lines 1-9. In addition, a server 12 coupled to the network 16 is described at page 11 lines 10-12, wherein the server 12 includes: an application code source 18 that stores a client application; and a server code manager 22 coupled to the application code source 18. At least at page 11 lines 13-15, an application code transformation manager 20 is coupled to the application code source 18, for transforming the client application from a first format to a native binary format compatible with a native instruction set of the CPU of the client.

At least at page 11 lines 22-24 and Figure 1, a server code segment manager 22 is coupled to the application code transformation manager 22, for parsing the client application in the native binary format into a plurality of code segments. At page 14 lines 12-17, the parsing of said code segments being dynamically performed based on actual server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis, at page 17 lines 2-6 and shown at least in Figure 2B block 48, the parsing is configured based on predicted code segment usage or prior code segment usage history, at least one of said plurality of code segments being transmitted to the client via the network.

In addition, at least at page 12 lines 7-9 and again at Figure 1, a client 14 is coupled to the network 16 said client 14 not having said client application stored thereon, wherein the client comprises: a CPU for natively executing at least one of said plurality of said code segments derived from the client application stored on said server 12. At page 12 lines 11-13, a code cache 30 is coupled to the CPU, for storing said code segments; and at page 12 lines 17-20 a client code manager 28 is coupled to the code cache 30 for launching the client application by requesting that the server code manager 22 transmit at least one of the plurality of dynamically tailored code segments to the client 14, receiving at least one of the dynamically tailored code segment from the server, storing the dynamically tailored code segment in the code cache 30, and at page 12 lines 24-26 and shown in Figure 1, executing at least one of the plurality of dynamically tailored code segments using the CPU until the executed dynamically tailored code segment attempts to pass control to a required code segment not stored in the code cache 30, at which point control passes back to the client code

manager 26 to retrieve the required code segment from the server 12, with the CPU continuing execution with the required code segment.

In Claim 9, a server 12 is described and shown at least in Figures 1 and 2B. For example, at Figure 1 and at least page 10 lines 25-27, the server 12 comprises a network interface that couples the server 12 to a network 16, which in turn is coupled to a client 14. In addition, at lines 1-4 of page 11 the server 12 of Figure 1 includes an application code source 18 that stores a client application a server code manager 22 coupled to the application code source 18 and the network interface. With respect to Figure 2B and page the server code manager 22 derives native binary code segments in a native execution format required by client CPUs from the application code source.

At page 14 lines 12-17, the deriving of the native binary code segments being dynamically performed based on actual server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis, and at page 17 lines 2-6 and shown at least in Figure 2B block 48, the deriving is configured based on predicted code segment usage or prior code segment usage history. At block 50 of Figure 2B and page 17 lines 8-12, the server 12 transmits at least one of the native binary code segments to said client not having said client application stored thereon upon requests received from said client.

In Claim 14, a client is described and shown at least in Figures 1 and 2A. For example, at Figure 1 and at least page 11 lines 25-27, a network interface that couples the client 14 to a network 16, which in turn is coupled to a server 12. In addition, at page 12 lines 1-7, a CPU for natively executing code segments derived from the client application stored on the server 12. At page 14 lines 12-17, the code segments derived from the client application are dynamically tailored by the client application based on actual server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis. At page 17 lines 2-6 the segments are configured based on predicted code segment usage or prior code segment usage history;

At page 12 lines 11-13, a client side code cache 30 is coupled to the CPU, for storing code segments; and at page 12 lines 17-20 a client code manager 28 is coupled to the code cache, wherein the client code manager 28 launches the client application stored on said server 12 and not on said client by requesting that the server transmit at least one of said code segments to the client 14, receiving at least one of said code segments from the server, storing at least one of said code segments in the code cache 30, and at page 12 lines 24-26 and shown in Figure 1, executing at least one of said code segments using the client CPU until the code

segment attempts to pass control to a required code segment not stored in the code cache 30, at which point control passes back to the client code manager 26 to retrieve the required code segment from the server 12, with the CPU continuing execution with the required code segment.

In Claim 18, method of executing an application in networked system is described and shown at least in Figures 1 and 2A-2B. For example, at Figure 1 and at least page 11 lines 25-27, at a client 14: issuing a code segment request to a server 12 coupled to the client 14 by a network 16. In addition, at Figure 1, at the server 12: receiving the code segment request from the client 14. At least at Figures 1 and 2B and at lines 1-4 of page 11 one embodiment derives a plurality of code segments, in a native execution format required by the client, from an application code source stored on said server and not on said client.

At page 14 lines 12-17, the code segments being dynamically tailored based on server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements. Further, at least at page 17 lines 2-6 and shown at least in Figure 2B block 48 the code segments are configured based on predicted code segment usage or prior code segment usage history.

As shown in Figure 2B and at least page 17 lines 5-7, transmitting at least one of the plurality of code segments to the client 14. As shown in Figure 2C, client 14 receives at least one of the plurality of code segments. Further, page 17 lines 22-26 provides adjusting branches in at least one of the plurality of code segments having targets not in a code cache of the client 14 to cause code segments containing the targets to be requested from the server 12. At block 44 of Figure 2A and page 18 lines 2-4, one embodiment emits at least one of the plurality of code segments into the code cache 30. At page 18 lines 5-6, block 32 of Figure 2A executes at least one of the plurality of code segments natively from the code cache 30.

In Claim 23, a computer program product having one computer usable medium having computer readable code embodied therein for causing an application to be executed in a networked system, the computer program product is described and shown at least in Figures 1 and 2A-2C. For example, at Figure 1 and at least page 11 lines 25-27 first computer readable program code devices configured to cause a client 14 to issue a code segment request to a server 12 coupled to the client 14 by a network 16.

In addition, at page 11 lines 9-13 and shown in Figures 1 and 2B, a second computer readable program code devices configured to cause the server 12 to receive the code segment request from the client 14, derive a plurality of code segments in a native execution format

required by the client 14 from an application code source stored on said server 12 and not on said client 14. At page 14 lines 12-17, the plurality of code segments are derived dynamically from said application code source based on server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis, and transmit at least one of said plurality of said code segments to the client 14.

At page 11 lines 9-13 and shown in Figures 1 and 2A and 2C, a third computer readable program code devices configured to cause the client to receive at least one of said plurality of said code segments, page 17 lines 22-26, adjust branches in at least one of said plurality of said code segments having targets not in a code cache of the client to cause code segments containing the targets to be requested from the serve. At block 44 of Figure 2A and page 18 lines 2-4, one embodiment emits at least one of said plurality of said code segments into the code cache. At page 18 lines 5-6, block 32 of Figure 2A executes at least one of said plurality of said code segments natively from the code cache.

VI. Grounds of Rejection to Be Reviewed on Appeal

1. Claims 2-24 stand rejected under 35 USC 103(a) as being unpatentable over Shimura (6,370,687) in view of Official Notice.

VII. Argument

1. Whether Claims 2-24 are unpatentable under 35 U.S.C. § 103(a) over Shimura in view of Official Notice.

Appellants respectfully submit that the rejection of the Claims is improper as the rejection of Claims 2-24 does not satisfy the requirements of a *prima facie* case of obviousness as claim features are not met by the cited references.

Appellants understand MPEP § 2141.02 (VI) to clearly provide: "prior art must be considered in its entirety, including disclosures that teach away from the claims" (emphasis added). Further, *In re Hedges,* 783 F.2d 1038, 228 USPQ 685 (Fed. Cir. 1986), [T]he totality of the prior art must be considered, and proceeding contrary to accepted wisdom in the art is evidence of nonobviousness" (emphasis added).

Further, a holding of obviousness can be based on a showing that there was "an apparent reason to combine the known elements in the fashion claimed." KSR, 127 S. Ct. at 1740-41, 82 USPQ2d at 1396. In other words, "there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness." Id., 127 S. Ct. at 1741, 82 USPQ2d at 1396 (quoting In re Kahn, 441 F.3d 977, 987, 78 USPQ2d 1329, 1336 (Fed. Cir. 2006)). However, this reasoning is not limited to the problem the patentee was trying to solve; "any need or problem known in the field of endeavor at the time of invention and addressed by the patent can provide a reason for combining the elements in the manner claimed," KSR, 127 S. Ct. at 1742,82 USPQ2d at 1397.

Regarding Independent Claims 2, 9, 14, 18 and 23, Appellants respectfully submit that Claim 2 (Claims 9, 14, 18 and 23 include similar features) includes the features "a server code segment manager coupled to the application code transformation manager, for parsing the client application in the native binary format into a plurality of code segments, said parsing of said code segments being dynamically performed based on actual server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis, and configured based on predicted code segment usage or prior code segment usage history" (emphasis added).

Appellants respectfully submit that Shimura does not teach this claimed feature. That is, Appellants do not understand Shimura to teach dynamic parsing of application code into code segments wherein the parsing of the code segments is dynamically performed based on

actual server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis (emphasis added).

For this reason, Appellants respectfully submit that the Examiner's rejections of the Claims are improper as the rejection of Claims 2-24 does not satisfy the requirements of a *prima facie* case of obviousness as claim features are not met by the cited reference. Accordingly, Appellants respectfully submit that the rejections of Claims 2-24 under 35 U.S.C. §103(a) is improper and should be reversed.

Response to Arguments

Regarding the response to Arguments, Appellants are submitting the following remarks in response. In these remarks, Appellants are addressing certain arguments presented in the Response to Remarks Section. While only certain arguments are addressed, this should not be construed that Appellants agree with the other arguments presented in the Response to Remarks Section.

Beginning on page 3 section 2 of the present Office Action, the feature of Claim 2 (Claims 9, 14, 18 and 23 include similar features) e.g., "dynamic parsing of application code into code segments wherein the parsing of the code segments is dynamically performed based on actual server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis" are discussed.

Specifically, the feature is addressed with the statement that "[T]he breadth of the limitation hinges upon the phrase "based on" which is clearly expansive." The Office Action then provides a plurality of examples that Appellants respectfully submit are not analogous with the Claimed features.

For example, on Page 4 lines 3-10 of the Present Office Action, the statement "[A]lternatively, if the server has no available execution overhead because it is operating at its capacity, it would be impossible for the server to also dynamically parse code segments" is improperly compared with the feature "dynamic parsing of application code into code segments wherein the parsing of the code segments is dynamically performed based on actual server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis."

Appellants respectfully submit that if, as stated, it is impossible for the server to also dynamically parse code segments, then the server cannot possibly be dynamically parsing. Thus, the Office Action itself provides no support that Shimura teaches dynamic parsing

based on actual server side execution overhead and instead simply provides an example of when parsing does not occur.

Further, Appellants respectfully submit that the responsive section 2 of the present Office Action repeats the same faulty argument when the "[C]lient has no available execution overhead" (page 4 line 8); when the "[n]etwork is inefficient and has no available bandwidth" (page 4 line 16); and when the "[c]lient has no available storage capacity" (page 4 last line).

Again, Appellants respectfully submit that not parsing is not analogous to dynamic parsing. In fact, Appellants submit that not parsing would be a different course of action than that taught by the present claimed features. That is, the present features do not teach "not parsing" but instead "dynamic parsing of application code into code segments wherein the parsing of the code segments is dynamically performed based on actual server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis" (emphasis added).

Thus, for these reasons, Appellants respectfully submit that the rejections of the Claims are improper as the rejection of Claims 2-24 does not satisfy the requirements of a *prima facie* case of obviousness as claim features are not met by the cited reference. Accordingly, Appellants respectfully submit that the rejection of Claims 2-24 under 35 U.S.C. §103(a) is improper and should be reversed.

Conclusion

Appellants believe that pending Claims 2-24 are directed toward patentable subject matter. As such, Appellants respectfully request that the rejection of Claims 2-24 be reversed. Appellants wishes to encourage the Examiner or a member of the Board of Patent Appeals to telephone the Appellant's undersigned representative if it is felt that a telephone conference could expedite prosecution.

Respectfully submitted,

Wagner Blecher LLP

Dated: 2/11/2008

/John P. Wagner, Jr./

John P. Wagner, Jr.
Registration No.: 35,398

Wagner Blecher LLP
Westridge Business Park
123 Westridge Drive
Watsonville, CA 95076

Phone: (408) 377-0500
Facsimile: (408) 722-2350

2. A networked system comprising:

a network;

a server coupled to the network, wherein the server includes:

an application code source that stores a client application; and

a server code manager coupled to the application code source;

an application code transformation manager coupled to the application code source, for transforming the client application from a first format to a native binary format compatible with a native instruction set of the CPU of the client; and

a server code segment manager coupled to the application code transformation manager, for parsing the client application in the native binary format into a plurality of code segments, said parsing of said code segments being dynamically performed based on actual server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis, and ~~is~~ configured based on predicted code segment usage or prior code segment usage history, at least one of said plurality of code segments being transmitted to the client via the network; and

a client coupled to the network said client not having said client application stored thereon, wherein the client comprises:

a CPU for natively executing at least one of said plurality of said code segments derived from the client application stored on said server;

a code cache coupled to the CPU, for storing said code segments; and

a client code manager coupled to the code cache for launching the client application by requesting that the server code manager transmit at least one of the plurality of dynamically tailored code segments to the client, receiving at least one of the dynamically tailored code segment from the server, storing the dynamically tailored code segment in the code cache, and executing at least one of the plurality of dynamically tailored code segments using the CPU until the executed dynamically tailored code segment attempts to pass control to a required code segment not stored in the code cache, at which point control passes back to the client code manager to retrieve the required code segment from the server, with the CPU continuing execution with the required code segment.

3. The networked system of claim 2 wherein the first format is a native binary format other than the native binary format of the CPU of the client, and the application code transformation manager comprises a transformation engine to transform the client application from the first format to the native binary format of the CPU of the client.

4. The networked system of claim 2 wherein the first format is a source code text format of a programming language, and the application code transformation manager comprises a compiler that compiles and links the client application into a native binary format of the CPU of the client.

5. The networked system of claim 2 wherein the first format is a virtual machine format, and the application code transformation manager comprises a just-in-time compiler that compiles and links the client application into a native binary format of the CPU of the client.

6. The networked system of claim 2 wherein the client code manager comprises:

a client code segment manager coupled to the network and the code cache, wherein the client code manager requests needed segments from the server; and

a code cache linker and manager coupled to the code cache and client code cache segment manager, wherein the code cache linker and manager links the code segment received from the server into the code cache, emits the received code segment into the code cache, and branches to the received code segment the code cache.

7. The networked system of claim 6 wherein the code cache linker and manager further comprise:

adjusting any branch targets in code segments stored in the code cache that need to branch to the received code segment and had previously been adjusted to branch out of the code cache to the client code segment manager to now branch to appropriate locations within the received code segment;

adjusting any branch instructions in the received code segment having branch targets that branch to code segments currently in the code cache to branch to the appropriate code segments in the code cache; and

adjusting any branch instructions in the received code segment having branch targets that need to branch to code segments not in the code cache to branch out of the code cache to the client code segment manager to request the code segment containing the branch targets.

8. The networked system of claim 6 wherein the code cache linker and manager includes a code cache maintenance unit coupled to the code cache, for removing old and unneeded code segments from the code cache, replacing older segments with newly received segments when the code cache reaches a certain threshold.

9. A server comprising:

a network interface that couples the server to a network, which in turn is coupled to a client;

an application code source that stores a client application; and

a server code manager coupled to the application code source and the network interface, wherein the server code manager derives native binary code segments in a native execution format required by client CPUs from the application code source, the deriving of the native binary code segments being dynamically performed based on actual server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis, and configured based on predicted code segment usage or prior code segment usage history, and transmits at least one of the native binary code segments to said client not having said client application stored thereon upon requests received from said client.

10. The server of claim 9 wherein the server code manager comprises:

an application code transformation manager coupled to the application code source, for transforming the client application from a first format to the native execution format; and a server code segment manager coupled to the application code transformation manager, for parsing the client application in the native execution format into code segments that are transmitted to said client via the network.

11. The server of claim 10 wherein the first format is a native execution format other than the native execution format required by a client CPU, and the application code transformation manager comprises a transformation engine to transform the client application from the first format to the native execution format required by the client CPU.

12. The server of claim 10 wherein the first format is a source code text format of a programming language, and the application code transformation manager comprises a compiler that compiles and links the client application into a native binary format required by a client CPU.

13. The server of claim 10 wherein the first format is a virtual machine format, and the 2 application code transformation manager comprises a just-in-time compiler that compiles and links the client application into a native binary format required by a client CPU.

14. A client comprising:

a network interface that couples the client to a network, which in turn is coupled to a server;

a CPU for natively executing code segments derived from the client application stored on the server, wherein the code segments derived from the client application are dynamically tailored by the client application based on actual server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis, and configured based on predicted code segment usage or prior code segment usage history;

a client side code cache coupled to the CPU, for storing code segments; and

a client code manager coupled to the code cache, wherein the client code manager launches the client application stored on said server and not on said client by requesting that the server transmit at least one of said code segments to the client, receiving at least one of said code segments from the server, storing at least one of said code segments in the code cache, and executing at least one of said code segments using the client CPU until the code segment attempts to pass control to a required code segment not stored in the code cache, at which point control passes back to the client code manager to retrieve the required code segment from the server, with the CPU continuing execution with the required code segment.

15. The client of claim 14 wherein the client code manager comprises:

a client code segment manager coupled to the network and the code cache, wherein the client code manager requests needed segments from the server; and

a code cache linker and manager coupled to the code cache and client code cache segment manager, wherein the code cache linker and manager links the code segment received from the server into the code cache, emits the received code segment into the code cache, and branches to the received code segment in the code cache.

16. The client of claim 15 wherein the code cache linker and manager further comprise:

adjusting any branch targets in code segments stored in the code cache that need to branch to the received code segment and had previously been adjusted to branch out of the code cache to the client code segment manager to now branch to appropriate locations within the received code segment;

adjusting any branch instructions in the received code segment having branch targets that branch to code segments currently in the code cache to branch to the appropriate code segments in the code cache; and

adjusting any branch instructions in the received code segment having branch targets that need to branch to code segments not in the code cache to branch out of the code cache to the client code segment manager to request the code segment containing the branch targets.

17. The client of claim 15 wherein the code cache linker and manager includes a code cache maintenance unit coupled to the code cache, for removing old and unneeded code segments

from the code cache, replacing older segments with newly received segments when the code cache reaches a certain threshold.

18. A method of executing an application in networked system comprising:

at a client: issuing a code segment request to a server coupled to the client by a network;

at the server: receiving the code segment request from the client;

deriving a plurality of code segments, in a native execution format required by the client, from an application code source stored on said server and not on said client, the code segments being dynamically tailored based on server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements, and is configured based on predicted code segment usage or prior code segment usage history;

transmitting at least one of the plurality of code segments to the client;

at the client: receiving at least one of the plurality of code segments;

adjusting branches in at least one of the plurality of code segments having targets not in a code cache of the client to cause code segments containing the targets to be requested from the server;

emitting at least one of the plurality of code segments into the code cache; and

executing at least one of the plurality of code segments natively from the code cache.

19. The method of claim 18 wherein deriving a code segment in a native execution format required by the client from an application code source comprises transforming the application source from a native execution format other than that required by the client into the native execution format required by the client.

20. The method of claim 18 wherein deriving a code segment in a native execution format required by the client from an application code source comprises compiling and linking the application code source from a code text format of a programming language into the native execution format required by the client.

21. The method of claim 18 wherein deriving a code segment in a native execution format required by the client from an application code source comprises using a just-in-time compiler to compile the application code source from a virtual machine format into the native execution format required by the client.

22. The method of claim 18 wherein the code segment is a first code segment, and executing the code segment natively from the code cache includes:

at the client: executing a branch in the first code segment that seeks to branch to a second code segment not in the code cache; and

issuing a code segment request for the second code segment to the server;

at the server: receiving the code segment for the second code segment from the client; deriving the second code segment in the native execution format required by the client from the application code source; and

transmitting the second code segment to the client; and

at the client: receiving the second code segment;

adjusting any branches in the first code segment stored in the code cache that need to branch to the second code segment and had previously been adjusted to cause the second code segment to be requested from the server to now branch to appropriate locations within the second code segment; adjusting any branches in the second code segment having targets in the first code segment in the code cache to branch to the appropriate location within the first code segment; adjusting any branches in the second code segment having branch targets in code segments not in the code cache to cause the code segments not in the code cache to be requested by the server; emitting the second code segment into a code cache; and

continuing execution in the second code segment at the location to which the first code segment attempted to branch.

23. A computer program product, comprising:

at least one computer usable medium having computer readable code embodied therein for causing an application to be executed in a networked system, the computer program product including:

first computer readable program code devices configured to cause a client to issue a code segment request to a server coupled to the client by a network;

second computer readable program code devices configured to cause the server to receive the code segment request from the client, derive a plurality of code segments in a native execution format required by the client from an application code source stored on said server and not on said client, wherein plurality of code segments are derived dynamically from said application code source based on server-side and client-side execution overhead, network bandwidth efficiency and client-side storage requirements on a per client basis, and transmit at least one of said plurality of said code segments to the client; and

third computer readable program code devices configured to cause the client to receive at least one of said plurality of said code segments, adjust branches in at least one of said plurality of said code segments having targets not in a code cache of the client to cause code segments containing the targets to be requested from the serve, emit at least one of said plurality of said code segments into the code cache, and execute at least one of said plurality of said code segments natively from the code cache.

24. The computer program product of claim 23 wherein the code segment is a first code segment, and the third computer readable program code devices includes:

fourth computer readable program code devices configured to cause the client to execute a branch in the first code segment that seeks to branch to a second code segment not in the code cache, and issue a code segment request for the second code segment to the server;

fifth computer readable program code devices configured to cause the server to receive the code segment for the second code segment from the client, derive the second code segment in the native execution format required by the client from the application code source, and transmit the second code segment to the client; and

sixth computer readable program code devices configured to cause the client to receive the second code segment, adjusting any branches in the first code segment stored in the code cache that need to branch to the second code segment and had previously been adjusted to cause the second code segment to be requested from the server to now branch to appropriate locations within the second code segment, adjust any branches in the second code segment having targets in the first code segment in the code cache to branch to the appropriate location within the first code segment, adjust any branches in the second code segment having branch targets in code segments not in the code cache to cause the code segments not in the code cache to be requested by the server, emit the second code segment into a code cache, and continue execution in the second code segment at the location to which the first code segment attempted to branch.

IX. Evidence Appendix

No evidence is herein appended.

X. Related Proceedings Appendix

No related proceedings.